

Automated Analog Circuit Synthesis using a Linear Representation

Jason D. Lohn¹ and Silvano P. Colombano²

¹ Caelum Research Corporation, NASA Ames Research Center,
Mail Stop 269-1, Moffett Field, CA 94035-1000, USA
email: jlohn@ptolemy.arc.nasa.gov

² Computational Sciences Division, NASA Ames Research Center,
Mail Stop 269-1, Moffett Field, CA 94035-1000, USA
email: scolombano@mail.arc.nasa.gov

Abstract. We present a method of evolving analog electronic circuits using a linear representation and a simple unfolding technique. While this representation excludes a large number of circuit topologies, it is capable of constructing many of the useful topologies seen in hand-designed circuits. Our system allows circuit size, circuit topology, and device values to be evolved. Using a parallel genetic algorithm we present initial results of our system as applied to two analog filter design problems. The modest computational requirements of our system suggest that the ability to evolve complex analog circuit representations in software is becoming more approachable on a single engineering workstation.

1 Introduction

Analog circuits are of great importance in electronic system design since the world is fundamentally analog in nature. While the amount of digital design activity far outpaces that of analog design, most digital systems require analog modules for interfacing to the external world. It was recently estimated that approximately 60% of CMOS-based application-specific integrated circuit (ASIC) designs incorporated analog circuits [1]. With challenging analog circuit design problems and fewer analog design engineers, there are economic reasons for automating the analog design process, especially time-to-market considerations.

Techniques for analog circuit design automation began appearing about two decades ago. These methods incorporated heuristics [13], knowledge-bases [4], and simulated annealing [11]. Efforts using techniques from evolutionary computation have appeared over the last few years. These include the use of genetic algorithms (GAs) [5] to select filter component sizes [6], to select filter topologies [3], and to design operational amplifiers using a small set of topologies [10]. The research of Koza and collaborators [8] on analog circuit synthesis by means of genetic programming (GP) is likely the most successful approach to date. Unlike previous systems, the component values, number of components, and the circuit topologies are evolved. The genetic programming system begins with minimal

knowledge of analog circuit design and creates circuits based on a novel circuit-encoding technique. Various analog filter design problems have been solved using genetic programming (e.g., [9]), and an overview of these techniques, including eight analog circuit synthesis problems, is found in [8]. A comparison of genetic-based techniques applied to filter design appears in [14] and work on evolving CMOS transistors for function approximation [12] has also recently appeared.

The system we present here was motivated by the genetic programming system described above. Our investigation centers on whether a linear representation and simple unfolding technique, coupled with modest computer resources, could be effective for evolving analog circuits. In the GP system, a hierarchical representation is manipulated by evolution, and a biologically-inspired encoding scheme is used to construct circuits. In our system we use a linear genome representation and a simple unfolding process to construct circuits. As mentioned, our current system is topology-constrained, yet such constraints were deemed reasonable since a vast number of circuit topologies are attainable. Our technique presented below differs from the previous GA techniques in that we allow both topology and component sizes to be evolved. In [14], a GA approach is presented in which topologies and component values are evolved for circuits containing up to 15 components. Here we use dynamically-sized representations in the GA so that circuits containing up to 100 components can be evolved. Using a cluster of six engineering workstations (1996 Sun Ultra), we present evolved circuit solutions to two filter design problems.

2 Linear Representation

Circuits are represented in the genetic algorithm as a list of bytecodes which are interpreted during a simple unfolding process. A fixed number of bytecodes represent each component as follows: the first is the opcode, and the next three represent the component value. Component value encoding is discussed first.

Using three bytes allows the component values to take on one of 256^3 values, a sufficiently fine-grained resolution. The raw numerical value of these bytes was then scaled into a reasonable range, depending on the type of component. Resistor values were scaled sigmoidally between 1 and 100K ohms using $1/(1 + \exp(-1.4(10x - 8)))$ so that roughly 75% of the resistor values were biased to be less than 10K ohms. Capacitor values were scaled between approximately 10 pF and 200 μ F and inductors between roughly 0.1 mH and 1.5 H.

The opcode is an instruction to execute during circuit construction. In the current design of our system, we use only “component placement” opcodes which accomplish placement of resistors, capacitors, and inductors. The five basic opcode types are: *x*-move-to-new, *x*-cast-to-previous, *x*-cast-to-ground, *x*-cast-input, *x*-cast-to-output, where *x* can be replaced by R (resistor), C (capacitor), or L (inductor). In a circuit design problem involving only inductors and capacitors (an LC circuit), ten opcodes would be available to construct circuits (five for capacitors and five for inductors).

The circuit is constructed between fixed input and output terminals as shown in Fig. 1. An ideal AC input voltage source v_s is connected to ground and to a source resistor R_s . The circuit's output voltage taken across a load resistor R_l .

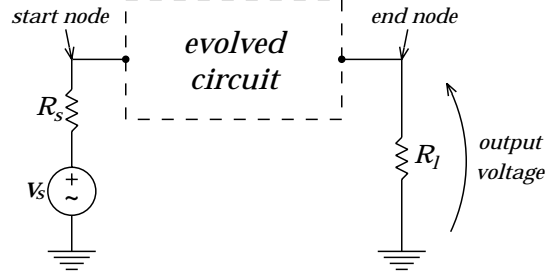


Fig. 1. Artificially evolved circuit is located between fixed input and output terminals (v_s is an ideal ac voltage source, R_s is the source resistance, R_l is the load resistance).

To construct the circuit, a “current node” register (abbreviated CN; with “current” used in the sense of present, not electrical current) is used and initialized to the circuit’s input node. The unfolding process then proceeds to interpret each opcode and associated component values, updating the CN register if necessary. The x -move-to-new opcode places one end of component x at the current node (specified by the CN register) and the other at a newly-created node. The CN register is then assigned the value of the newly-created node. The “ x -cast-to-” opcodes place one end of component x at the current node and the other at either the ground, input, output, or previously-created node. After executing these opcodes, the CN register remains unchanged. The meanings of each opcode are summarized in Table 1. All five opcode types place components into the circuit, although they could be designed to do other actions as well, e.g., move without placement.

Opcode	Destination Node	CN Register
x -move-to-new	newly-created node	assigned the newly-created node
x -cast-to-previous	previous node	unchanged
x -cast-to-ground	ground node	unchanged
x -cast-to-input	input node	unchanged
x -cast-to-output	output node	unchanged

Table 1. Summary of opcode types used in current system. x denotes a resistor, capacitor, or inductor.

The list of bytecodes is a variable-length list (the length is evolved by the GA). Thus, circuits of various sizes are constructed. When the decoding process

reaches the last component to place in the circuit, we arbitrarily chose to have the last node (value in CN) connected to the output terminal by a wire. By doing so, we eliminate unconnected branches.

We had two goals in designing the above encoding scheme. First, we wanted to see if a very simple set of primitives encoded in a linear fashion could indeed be used to successfully evolve circuits. Second, we wanted to minimize computer time during the genetic algorithm run. By keeping the decoding process minimal, the total time for fitness evaluations is thus reduced. Along the same lines, we wanted to keep circuit “repair” operations (e.g., removal of unconnected nodes) to a minimum since these also slow the system down.

The most significant restriction of our technique is that it cannot support all possible circuit topologies: circuit branches off of the main “constructing thread” cannot, in general, contain more than one node (there are some exceptions to this). The constructing thread is the sequence of components that are created by the *x*-move-to-new opcode. The constructing thread itself can be of varying lengths and can contain both series and parallel configurations. In spite of these limitations, our system allows creation of circuits with a large variety of topologies, especially topologies seen in hand-designed circuits (e.g., ladder constructs). We have lessened the topology restrictions somewhat by allowing “move-to” opcodes and will report on these efforts in the future.

3 Genetic Algorithm

The genetic algorithm operates on a population of dynamically-sized bytecode arrays. In practice we imposed a maximum size of about 400 bytes (100 circuit components) in order to accommodate population sizes of up to 18,000 individuals in our GA runs. The crossover and mutation (per locus) rates were set at 0.8 and 0.2 respectively. An overview of the evaluation process is depicted in Fig. 2. As in the GP system mentioned above, we used the Berkeley SPICE circuit simulation program to simulate our circuits. The array of bytecodes was interpreted in the manner previously described, and resulted in a SPICE netlist representation. The netlist is processed by SPICE and the output is then used to compute fitness for the individual. Fitness was calculated as the absolute value of the difference of the individual’s output and the target output. These error values were summed across evaluation points, with error being the distance between the target and the value the individual produced.

The parallel genetic algorithm implemented uses master/slave style parallelism [2] over a network of UNIX-based computers. A controlling host computer performs GA functions and distributes a population of bytecoded-individuals to specified number of worker nodes using socket connections. The worker nodes decode the individuals into SPICE netlists which are then fed into SPICE via FIFO pipes to minimize disk activity. Fitness is calculated using SPICE’s output, and then sent back to the host. Hundreds of individuals (and fitness scores) are packaged into a single message so that external network congestion delays are minimized. The SPICE program itself required little modification since it runs

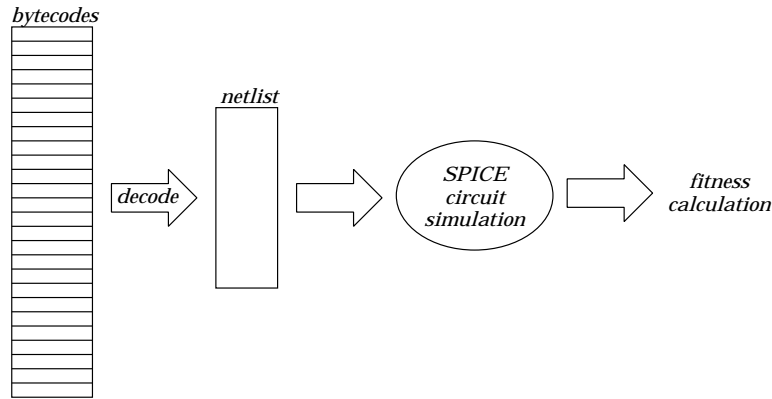


Fig. 2. Overview of circuit evaluation process starting with bytecoded representation and ending with fitness score.

as a separate process. Written in the C programming language, the system currently runs on Sun workstations and is portable to other UNIX systems (e.g., we have ported the software to PCs running UNIX). This allows the system to run on UNIX-based clusters comprised of computers from different manufacturers.

4 Experimental Results

We attempted to evolve two analog filter circuits. The choice of using passive analog filters was inspired by the previous studies and is a good choice for testing the effectiveness of our system for three reasons. First, all components have two-terminals, the minimum number possible. If the proposed system could not evolve useful circuits using two-terminaled devices, then attempting to evolve circuits using more complex components (e.g., transistors) would likely prove ineffective. Second, there are no energy sources required within the circuit which further reduces the complexity. Lastly, filter design is a well-understood discipline within circuit design. Its “design space” has been greatly explored [7] which allows us to compare our evolved designs to well-known designs.

The problems we present below are both low-pass filters. A low-pass filter is a circuit that allows low frequencies to pass through it, but stops high frequencies from doing so. In other words, it “filters out” frequencies above a specified frequency. The unshaded area in Fig. 3 depicts the region of operation for low-pass filters. Below the frequency f_p the input signal is passed to the output, potentially reduced (attenuated) by K_p decibels (dB). This region is known as the passband. Above the frequency f_s , the input signal is markedly decreased by K_s decibels. As labeled, this region is called the stopband. Between the passband and stopband the frequency response curve transitions from low to high attenuation. The parameter located in this region, f_c , is known as the cutoff frequency.

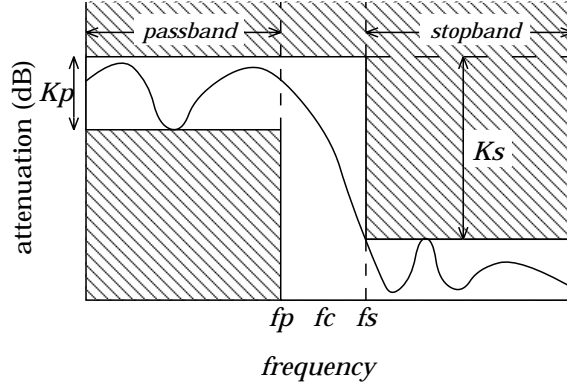


Fig. 3. Low-pass filter terminology and specifications. The crosshatched regions represent out-of-specification areas. An example frequency response curve that meets specifications is shown.

4.1 Electronic Stethoscope Circuit

The first circuit we attempted to evolve is one that is suitable for use in an electronic stethoscope. In this application, it is desired to filter out the extraneous high-frequency sounds picked up by a microphone which make it difficult to listen to (low-frequency) bodily sounds (e.g., a heart beating). As such, the frequency response specifications do not need to be extremely accurate since we are dealing with audible frequencies and the human ear cannot discern frequencies that are close together. The target frequency response data was taken from an actual electronic stethoscope, which was built with a cutoff frequency of 796 Hz corresponding to an output voltage of approximately 1 volt. This circuit is relatively easy to design and so we chose it as our first problem to solve.

The GA was allowed to use resistors and capacitors during evolution, resulting in an RC low-pass filter. The evolved circuit is shown in Fig. 4 and its frequency response, which matches almost exactly the target is shown in Fig. 5.

4.2 Butterworth Low-pass Filter

The second low-pass filter we evolved was more difficult. We chose a circuit that can be built using a 3rd-order Butterworth filter [7]. The specifications are as follows:

$$\begin{array}{ll} f_p = 925 \text{ Hz} & K_p = 3.0103 \text{ dB} \\ f_s = 3200 \text{ Hz} & K_s = 22 \text{ dB} \end{array}$$

Such a filter design can be derived using a ladder structure and component values found in published tables. The GA was allowed to use capacitors and inductors during evolution, resulting in an LC low-pass filter. The evolved circuit that meets these specifications is shown in Fig. 6 and its frequency response is

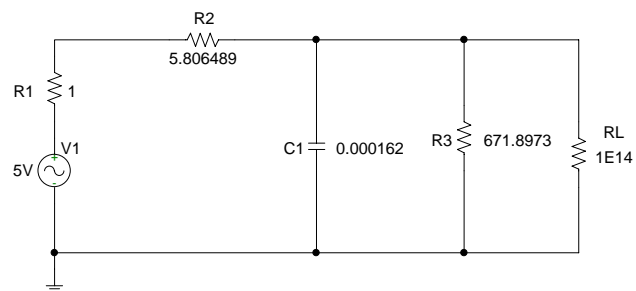


Fig. 4. Evolved low-pass filter for use in an electronic stethoscope (units are ohms and farads).

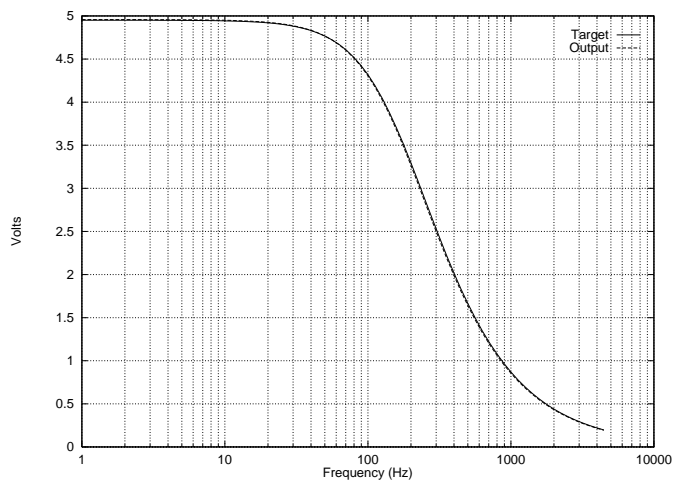


Fig. 5. Nearly identical frequency response curves for evolved and actual electronic stethoscope circuit. The frequency axis is scaled logarithmically.

shown in Fig. 7. It was found in generation 22 of a GA run that lasted approximately four hours using six Sun Ultra workstations working in parallel.

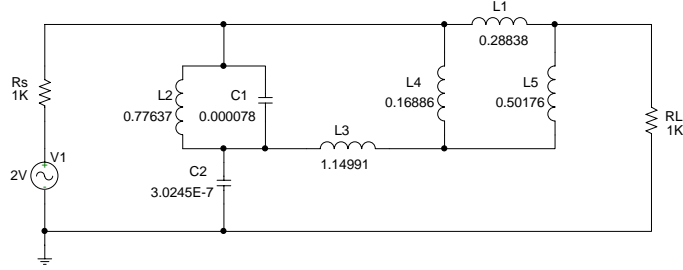


Fig. 6. Evolved 3rd-order Butterworth low-pass filter (units are ohms, farads, and henries).

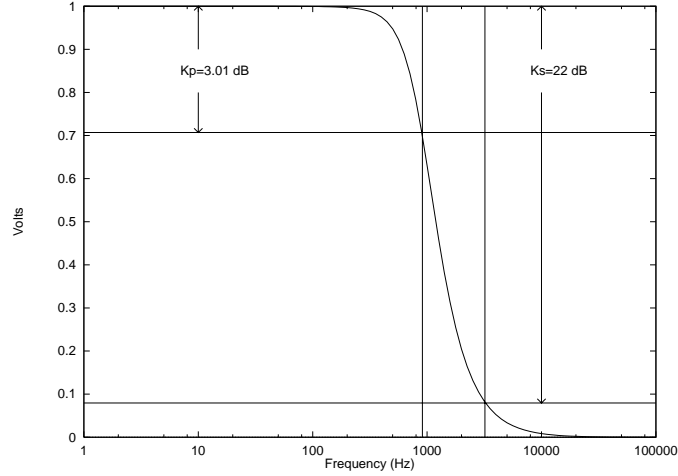


Fig. 7. Frequency response curve for evolved 3rd-order Butterworth low-pass filter. Attenuation specifications are also shown. The frequency axis is a scaled logarithmically.

5 Discussion

We have shown that a genetic algorithm using a simple linear circuit representation is capable of evolving two circuits of low to medium difficulty. The circuit construction method devised uses a very simple set of primitives encoded in a

linear fashion. Such a method helps to minimize the computer time required to evolve circuits by keeping the decoding and repairing processes shorter. Although this technique is topology-limited, the ability of our system to produce useful circuits was demonstrated. It is likely that these topological space restrictions are favorable to many filter designs, especially filters that are known to have less complex branching patterns (e.g., ladder structures). We intend to build upon this technique to allow for greater topologies and three-terminal devices such as transistors. With the previous successes in evolving analog circuits, and the encouraging early results of our system, we are optimistic that a subset of analog circuit design tasks may be routinely accomplished by means of evolutionary computation in the future.

6 Acknowledgments

The authors would like to thank M. Lohn, D. Stassinopoulos, G. Haith, and the anonymous reviewers for their helpful suggestions and comments.

References

1. G. Gielen, W. Sansen, *Symbolic Analysis for Automated Design of Analog Integrated Circuits*, Boston, MA: Kluwer, 1991.
2. D.E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison-Wesley, Reading, Mass, 1989.
3. J.B. Grimbleby, "Automatic Analogue Network Synthesis using Genetic Algorithms," *Proc. First Int. Conf. Genetic Algorithms in Engineering Systems: Innovations and Applications (GALESIA)*, 1995, pp. 53-58.
4. R. Harjani, R.A. Rutenbar, L.R. Carey, "A Prototype Framework for Knowledge-Based Analog Circuit Synthesis," *Proc. 24th Design Automation Conf.*, 1987.
5. J.H. Holland, *Adaptation in Natural and Artificial Systems*, Univ. of Michigan Press, Ann Arbor, 1975.
6. D.H. Horrocks, Y.M.A. Khalifa, "Genetically Derived Filters using Preferred Value Components," *Proc. IEE Colloq. on Linear Analogue Circuits and Systems*, Oxford, UK, 1994.
7. L.P. Huelsman, *Active and Passive Analog Filter Design*, New York: McGraw-Hill, 1993.
8. J.R. Koza, F.H. Bennett, D. Andre, M.A. Keane, F. Dunlap, "Automated Synthesis of Analog Electrical Circuits by Means of Genetic Programming," *IEEE Trans. on Evolutionary Computation*, vol. 1, no. 2, July, 1997, pp. 109-128.
9. J.R. Koza, F.H. Bennett, J.D. Lohn, F. Dunlap, M.A. Keane, D. Andre, "Use of Architecture-Altering Operations to Dynamically Adapt a Three-Way Analog Source Identification Circuit to Accommodate a New Source," in *Genetic Programming 1997 Conference*, J.R. Koza, K. Deb, M. Dorigo, D.B. Fogel, M. Garzon, H. Iba, and R.L. Riolo, (eds), Morgan Kaufmann, 1997, pp. 213-221.
10. M.W. Kruiskamp, *Analog Design Automation using Genetic Algorithms and Polytopes*, Ph.D. Thesis, Dept. of Elect. Engr., Eindhoven University of Technology, Eindhoven, The Netherlands, 1996.

11. E.S. Ochotta, R.A. Rutenbar, L.R. Carley, "Synthesis of High-Performance Analog Circuits in ASTRX/OBLX," *IEEE Trans. Computer-Aided Design*, vol. 15, pp. 273–294, 1996.
12. A. Stoica, "On Hardware Evolvability and Levels of Granularity," *Proc. 1997 Int. Conf. Intell. Systems and Semiotics*, 1997, pp. 244–247.
13. G.J. Sussman, R.M. Stallman, "Heuristic Techniques in Computer-Aided Circuit Analysis," *IEEE Trans. Circuits and Systems*, vol. 22, 1975.
14. R.S. Zebulum, M.A. Pacheco, M. Vellasco, "Comparison of Different Evolutionary Methodologies Applied to Electronic Filter Design," *1998 IEEE Int. Conf. on Evolutionary Computation*, Piscataway, NJ: IEEE Press, 1998, pp. 434–439.